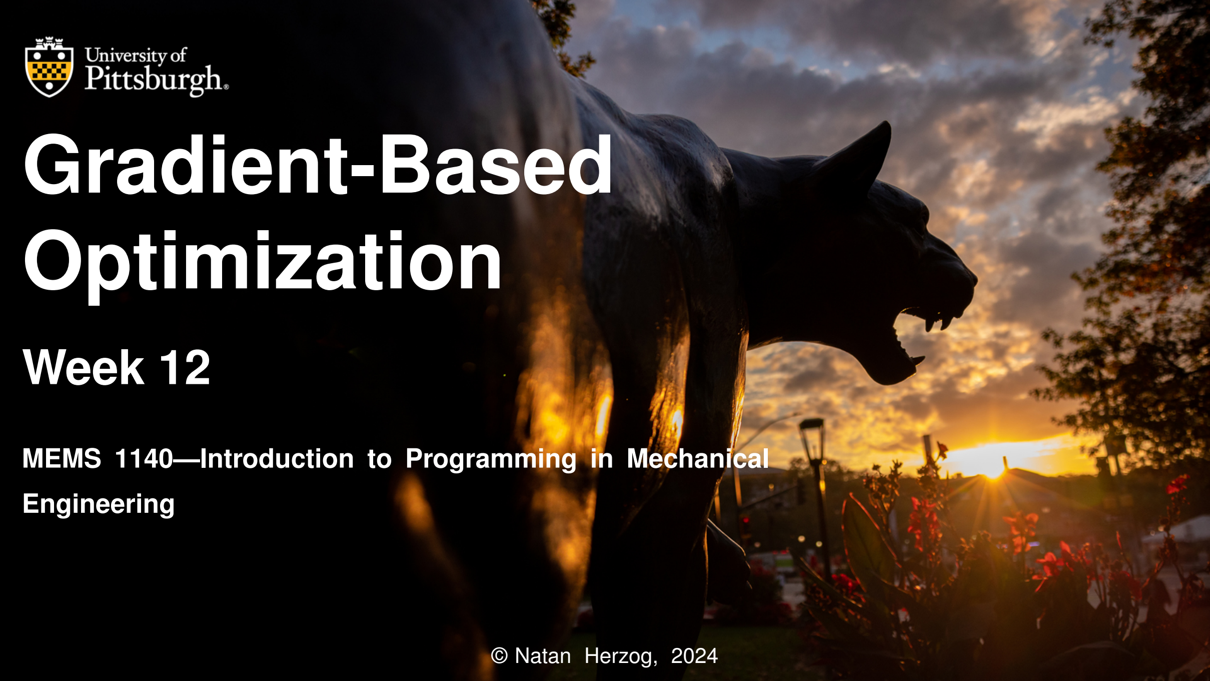


University of Pittsburgh

Gradient-Based Optimization

Week 12

MEMS 1140—Introduction to Programming in Mechanical Engineering

Learning Objectives (L.O.)

At the end of this lecture, you should understand/be able to:

- ❑ Advantages of gradient-based optimization over parametric sweep;
- ❑ How gradient-based optimization works;
- ❑ Implement gradient ascent for a simple unimodal surface;
- ❑ Limitations of gradient-based optimization for multimodal functions;

Table of Contents (ToC)

1 – Defining a Parametric Domain

Last lecture covered optimization by parametric sweep.

Recall that the whole method rests upon testing the entire domain for each parameter.

This requires either *knowing* that there is an extremum within some small region, just not the exact coordinates ...

... or casting a large enough domain to confidently catch the extremum.

1 – The Domain Problem

But of course, this comes with a tangible computational cost for testing larger domains.

And this becomes especially relevant for multi-parameter problems, which require sweeping *several* large domains.

Gradient-based optimization addresses this by algorithmically converging towards an extremum in comparably few steps.

2 – So How Does it Work?

At a very high level: Follow the direction of the steepest slope.

We already have the perfect tool for this: The **gradient**, $\vec{\nabla} f$, denotes the steepest slope of the surface f.

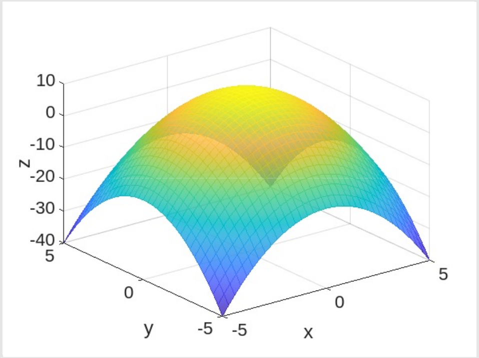
Recall the simplest case: $\vec{\nabla} f(x) = \dfrac{df(x)}{dx}$.

To optimize a function, take consecutive small steps along $\vec{\nabla} f$ to iteratively approach a maximum or minimum.

2 – Simple Surface Optimization

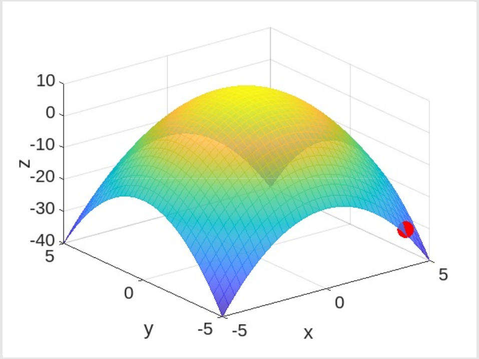
✓ L.O.1
➩ L.O.2
❏ L.O.3
❏ L.O.4

Consider the following surface: $f(x, y) = -(x^2 + y^2) + 10$



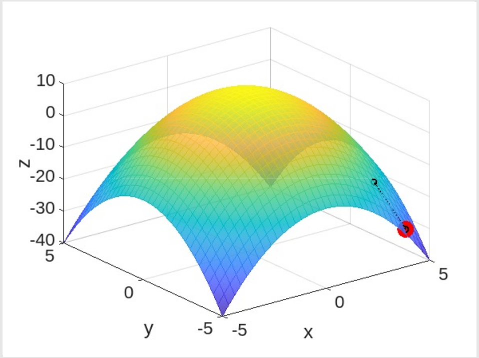
2 – Simple Surface Optimization

Let's start with a random starting point:



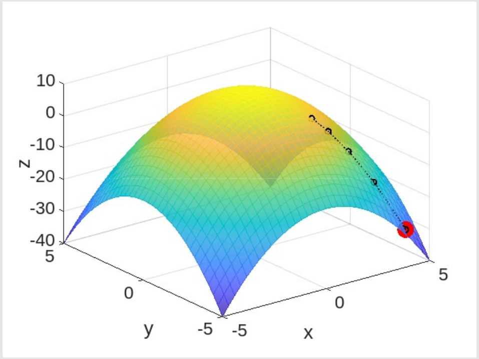
2 – Simple Surface Optimization

Then, after evaluating $\vec{\nabla} f$, take one step along that direction:



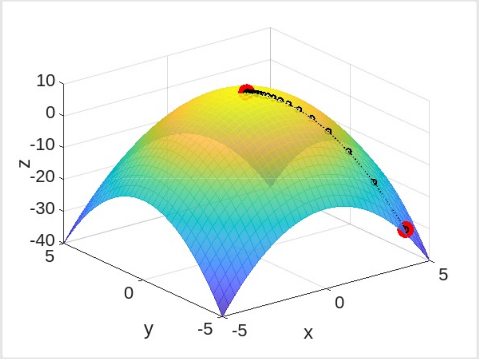
2 – Simple Surface Optimization

After a few more steps, we can see it start to approach the top:



2 – Simple Surface Optimization

We can skip the rest of the steps to see it converges to the top:

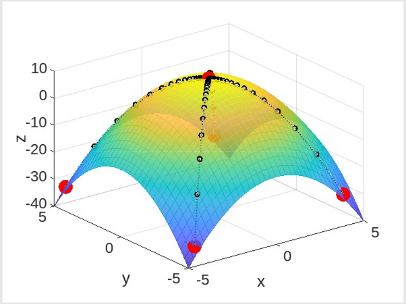


2 – General Algorithm Behavior

By following the gradient, the algorithm systematically approaches the global maximum of the function.

We reach the same end result regardless of the start location.

Note that the surface shown is just a visual aid.



2 – Following the Gradient

What does it mean to “take a step” along $\vec{\nabla} f(\vec{x})$?

The input coordinates, $\vec{x}$, get updated with each iteration according to the following formula:

Gradient **Ascent**:

$$\vec{x}_i = \vec{x}_{i-1} + \alpha \, \vec{\nabla} f(\vec{x}) \Big|_{\vec{x}_{i-1}}$$

Gradient **Descent**:

$$\vec{x}_i = \vec{x}_{i-1} - \alpha \, \vec{\nabla} f(\vec{x}) \Big|_{\vec{x}_{i-1}}$$

2 – General Algorithm Steps

After defining a start location, iteratively update the coordinates according to the formula before.

Note that α affects the overall step size. This is a tunable parameter, but we will mostly ignore it for simplicity.

Upon updating $\vec{x}$, evaluate $||\vec{x}_i - \vec{x}_{i-1}||$.

Continue this until $||\vec{x}_i - \vec{x}_{i-1}||$ is **less than** some convergence threshold. We will use the MATLAB built-in `eps`.

3 – Implementation: Define $f(x, y)$

```
f = @(x,y) -(x.^2 + y.^2) + 10; % function from previous example
.
.
.
.
.
.
.
.
.
.
.
```

3 – Implementation: Define $\vec{\nabla} f$

```
f = @(x,y) -(x.^2 + y.^2) + 10;
grad_f = @(x,y) [-2*x, -2*y];    % gradient of f(x,y)
.
.
.
.
.
.
.
.
.
.
```

3 – Implementation: Initialize Things

```
f = @(x,y) -(x.^2 + y.^2) + 10;
grad_f = @(x,y) [-2*x, -2*y];
x = 4.5; y = -4.8; delta = inf; z = []; pos = []; alpha = 0.1;
.
.
.
.
.
.
.
.
.
```

3 – Implementation: Set up the Loop

```
f = @(x,y) -(x.^2 + y.^2) + 10;
grad_f = @(x,y) [-2*x, -2*y];
x = 4.5; y = -4.8; delta = inf; z = []; pos = []; alpha = 0.1;
while delta > eps   % eps is MATLAB's floating-point relative accuracy
   .
   .
   .
   .
end
.
.
.
```

3 – Implementation: Evaluate $\vec{\nabla} f$

```
f = @(x,y) -(x.^2 + y.^2) + 10;
grad_f = @(x,y) [-2*x, -2*y];
x = 4.5; y = -4.8; delta = inf; z = []; pos = []; alpha = 0.1;
while delta > eps
  grad = grad_f(x, y);  % use the anonymous function defined previously
  .
  .
  .
end
.
.
.
```

3 – Implementation: Update $\vec{x}$

```
f = @(x,y) -(x.^2 + y.^2) + 10;
grad_f = @(x,y) [-2*x, -2*y];
x = 4.5; y = -4.8; delta = inf; z = []; pos = []; alpha = 0.1;
while delta > eps
  grad = grad_f(x, y);
  x = x + alpha*grad(1); y = y + alpha*grad(2); % update x and y
   .
   .
end
.
.
.
```

3 – Implementation: Evaluate $||\vec{x}_i - \vec{x}_{i-1}||$

```
f = @(x,y) -(x.^2 + y.^2) + 10;
grad_f = @(x,y) [-2*x, -2*y];
x = 4.5; y = -4.8; delta = inf; z = []; pos = []; alpha = 0.1;
while delta > eps
  grad = grad_f(x, y);
  x = x + alpha*grad(1); y = y + alpha*grad(2);
  delta_x = x - pos(end,1); delta_y = y - pos(end,2); % delta x and y
  delta = norm([delta_x, delta_y]);                   % ||x_i - x_i-1||
end
.
.
.
```

3 – Implementation: Print Results

```
f = @(x,y) -(x.^2 + y.^2) + 10;
grad_f = @(x,y) [-2*x, -2*y];
x = 4.5; y = -4.8; delta = inf; z = []; pos = []; alpha = 0.1;
while delta > eps
  pos(end+1,:) = [x,y]; z(end+1) = f(x, y); grad = grad_f(x, y);
  x = x + alpha*grad(1); y = y + alpha*grad(2);
  delta_x = x - pos(end,1); delta_y = y - pos(end,2);
  delta = norm([delta_x, delta_y]);
end
fprintf('Optimal Point: [%.2f,%.2f,%.2f]\n', ...
  pos(end,1),pos(end,2),z(end))                    % print results
fprintf('Reached in %d steps\n', length(z))
```

3 – Optimization Output

This relatively small bit of code is enough to fully define a gradient-descent algorithm.

For our example, we see that starting at $(4.5, -4.8)$ converges with the following output:

```
Optimal Point: [0.00,0.00,10.00]
Reached in 164 steps
```

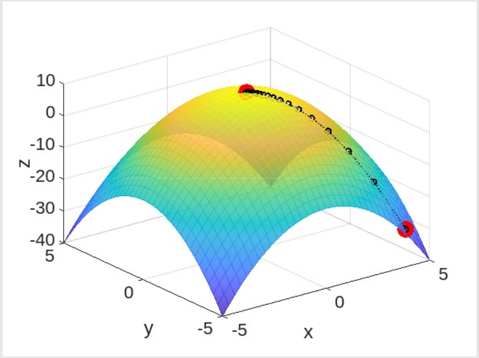
Command Window Output

3 – Implementation: Plotting

```
fsurf(f, 5*[-1,1,-1,1], 'FaceAlpha', 0.8, 'EdgeColor', 'interp');
xlabel('x'); ylabel('y'); zlabel('z');
set(gca, 'FontSize', 16)
set(gcf,'PaperUnits','centimeters','PaperSize',[10 7.5], ...
        'PaperPosition',[0 0 10 7.5])
hold on
points_graph = plot3(pos(:,1), pos(:,2), z);
start_point = scatter3(pos(1,1), pos(1,2), z(1), 150, 'filled');
end_point = scatter3(pos(end,1), pos(end,2), z(end), 150, 'filled');
points_graph.Marker = 'o'; points_graph.MarkerSize = 5;
points_graph.LineStyle = ':'; points_graph.LineWidth = 1.5;
points_graph.Color = 'k';
start_point.MarkerFaceColor = 'r'; end_point.MarkerFaceColor = 'r';
```

3 – Results of Plotting

This results in the figure we saw before:



4 – Multimodality

The surface from the previous example is considered “unimodal” because it only has one maximum point.

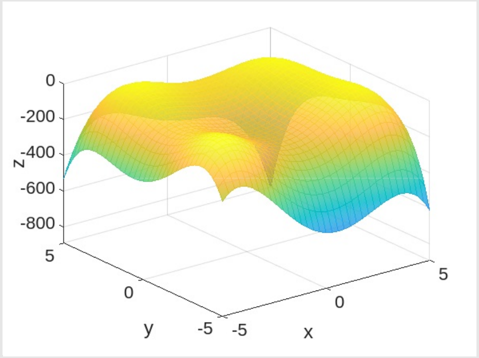
“Multimodal” functions have more than one extremum.

Such functions might have several local extrema, even if it only has one global extremum.

The simple implementation we covered in this lecture cannot reliably find the global extremum for multimodal functions.

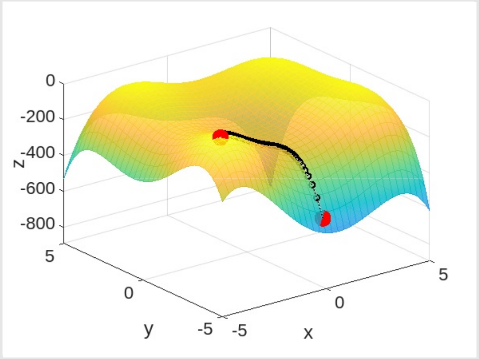
4 – Multimodal Surface

To illustrate this, consider the following surface with 4 maxima:



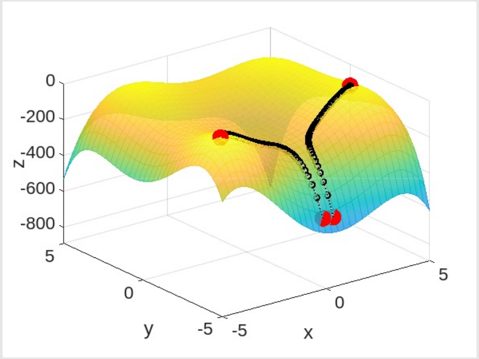
4 – Multimodal Surface

Starting at $(0, -4.8) \rightarrow 1^{\text{st}}$ maximum: $(-3.78, -3.28, 0)$



4 – Multimodal Surface

Starting at $(0.5, -4.8) \rightarrow$ 2nd maximum: $(3.58, -1.85, 0)$



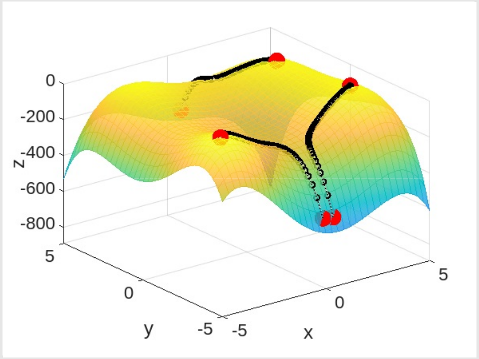
University of Pittsburgh

PITT | SWANSON ENGINEERING
MECHANICAL & MATERIALS SCIENCE

4 – Multimodal Surface

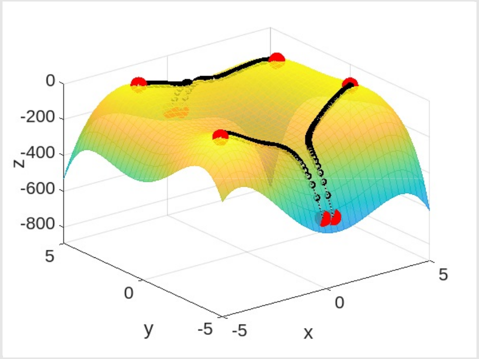
✓ L.O.1
✓ L.O.2
✓ L.O.3
⇨ L.O.4

Starting at $(0, 4.8) \rightarrow$ 3rd maximum: $(3, 2, 0)$



4 – Multimodal Surface

Starting at $(0.5, 4.8) \rightarrow$ 4th maximum: $(-2.81, 3.13, 0)$



5 – Concluding Thoughts

✓ L.O.1
✓ L.O.2
✓ L.O.3
✓ L.O.4

Optimization methods constitute a whole field of study unto themselves.

The past two lectures have barely scratched the surface.

"Optimization" is one of the most broadly applicable tools out there, especially for data-informed modeling in engineering.

You now have an *introduction* to the subject and I encourage you to explore it more deeply on your own.

6 – Summary

✓ L.O.1
✓ L.O.2
✓ L.O.3
✓ L.O.4

This lecture covered:

✓ The advantages of gradient-based optimization over parametric sweep

We avoid having to evaluate an entire domain, which is especially advantageous when trying to optimize multiple parameters.

✓ How gradient-based optimization works

By following the gradient of a surface in small iterative steps, the algorithm converges towards an extremum.

6 – Summary

✓ L.O.1

✓ L.O.2

✓ L.O.3

✓ L.O.4

✓ How to implement gradient-ascent for a simple surface

After defining a function to evaluate the gradient of a surface, a `while` loop iteratively updates the test coordinates until the change in coordinates decreases below the numerical threshold.

✓ The limitations of gradient-based optimization for multimodal problems

Gradient-based optimization from this lecture fails to obtain the global extremum for a surface that has multiple local extrema. It might converge differently depending on the starting conditions.